## Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## Lab 7 – Data Structures

The purpose of this lab is to give you more experience with using data structures in Java. There are two parts of this lab; the first part is required and the second is optional and is worth 2 bonus points, for a maximum possible grade of 12/10 on this lab. Each part has starter code in today’s downloads.

**Part 1 – Expression Balancing with Stacks (10 points)**

Using the Stack class, create a program, Balancer.java, that takes a mathematical expression, stored in a hard-coded String, and determines if the expression is balanced. For example, let’s look at the Quadratic Formula:

-(b \* b – ( 4 \* a \* c ) ) / (2 \* a)

**1 2 1 0 1 0**

You should increment a counter when you see a “(“ and decrement it when you see a “)”. The counter should never go negative and should be zero at the end of the expression, if the expression is balanced.

However, when equations are written, we typically use (, {, and [. Such as:

-{ [ b \* b – (4 \* a \* c) ] / (2 \* a) }

Use the following pseudocode to determine if the expression is balanced:

When you see an opening character, push it on the stack.

When you see a closing character, pop the stack. If the character popped off the stack doesn’t match the closing character (or there is nothing on the stack to pop), the expression is unbalanced.

If the stack is empty after reading through the expression, the expression is balanced. Otherwise, the expression is unbalanced.

**Part 2 – Priority Queues (Optional: 2 bonus points)**

We’ve seen the Queue class that allows users to **add** items to the tail of the queue and **remove** items from the head of the queue. Priority queues allow items to be added to the queue in the same manner, but they are removed based upon the *priority* of the item in the queue, using a method called **poll**. The item in the queue with the *lowest* priority is removed first.

For an item to be used by a PriorityQueue object, the item’s object type must implement Comparable using Comparable<Object>. In addition, the class must implement a method called compareTo that takes an object of that type as an argument so that the relative priority of each item in the priority queue can be determined.

Create a class called PQ.java that will instantiate a PriorityQueue of type Task. You will add 5 tasks to the PriorityQueue whose task names will be Task 1, Task 2, Task 3, Task 4, and Task 5 and their priorities will be 4, 7, 1, 3, and 6, respectively.

Once all the Task objects have been added, empty the PriorityQueue one item at a time and print out each item as it exits the priority queue.

The Task class (use Task.java for this) will contain a String called taskName and an int called myPriority. Its compareTo method will **compare** the priority of the current object versus another Task’s priority. You should also implement a toString method to neatly print out the contents of the Task object.